025921-22T

USE OF EDGE-BASED FINITE ELEMENTS FOR SOLVING THREE DIMENSIONAL SCATTERING PROBLEMS

A. Chatterjee
J.M. Jin
J.L. Volakis

NASA-Ames Research Center
Moffet Field, CA 94035
Grant NAG-2-541

Pacific Missile Test Center
Pt. Mugu, CA 03042-5000

Technical Report 025921-22T

August 1991

THE UNIVERSITY OF MICHIGAN

Radiation Laboratory
Department of Electrical Engineering and Computer Science
Ann Arbor, Michigan 48109-2122
USA

TECHNICAL REPORT

FOR

NASA Grant NAG-2-541

NASA Technical Monitor: Alex Woo

Grant Title: Use of Edge-Based Finite Elements for Solving Three Dimensional Scattering Problems

Institution: The Radiation Laboratory
Department of Electrical Engineering
and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122

Period Covered: February 1991 - September 1991

Report Authors: A. Chatterjee, J.M. Jin and J.L. Volakis

Principal Investigator: John L. Volakis
Telephone: (313) 764-0500

Table of Contents

Use of Edge-Based Finite Elements for Solving
Three Dimensional Scattering Problems

USE OF EDGE-BASED FINITE ELEMENTS FOR SOLVING THREE DIMENSIONAL SCATTERING PROBLEMS

A. Chatterjee, J.M. Jin and J.L. Volakis
Radiation Laboratory
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109-2122

Abstract

Edge-based finite elements are free from the drawbacks associated with node-based vectorial finite elements and are, therefore, ideal for solving three-dimensional scattering problems. The finite element discretization using edge elements is checked by solving for the resonant frequencies of a closed inhomogeneously filled metallic cavity. Significant improvements in accuracy are observed when compared to the classical node-based approach with no penalty in terms of computational time and with the expected absence of 'spurious' modes. A performance comparison between the edge-based tetrahedra and rectangular brick elements is carried out and tetrahedral elements are found to be more accurate than rectangular bricks for a given storage intensity. A detailed formulation for the scattering problem with various approaches for terminating the finite element mesh is also presented.

1 Introduction

The problem of computing the electromagnetic field scattered by a material body, when a field is incident on it, has great theoretical and practical importance. It is, usually, desirable to relate the characteristics of the scattered field in the far zone with the shape of the scatterer. Finite elements can be used to model complicated geometries but are of no help in computing the far field. The far field effect is thus represented in finite element analysis by enclosing the target within a fictitious boundary and enforcing a special boundary condition, often called the absorbing boundary condition, on the outer boundary. In this report, we discuss the choice of finite elements in the inner region. As a first step, we solve Maxwell's equations for the resonances of a closed cavity to verify the finite element discretization and decide on the type of basis functions and elements to use.

The occurrence of 'spurious' modes[1] in the node-based finite element approach has plagued the computation of cavity eigenvalues for years. This difficulty can be circumvented with the introduction of a penalty term[2] to render the finite element vector field solutions non-divergent. However, it is difficult to satisfy continuity requirements across material interfaces[3] or for geometries with sharp edges[4] using classical finite-elements, obtained by interpolating the nodal values of the vector field components. This is especially critical in solving scattering problems using classical finite elements where special techniques need to be used for satisfying normal and tangential continuity of the electric displacement and the electric field, respectively[5]. Edge elements, a type of vector finite element with their degrees of freedom associated with the edges of the mesh, have been shown to be free of these shortcomings[6],[7]. Edge-based finite elements are, therefore, a natural choice for treating three dimensional geometries. Generally these lead to more unknowns but the higher variable count is balanced by the greater sparsity of the finite element matrix so that the computation time required to solve such a system iteratively with a given accuracy is less than the traditional approach[8].

In this report, we have discussed the formulation for both the cavity and the scattering problems but have presented results only for the former. We have solved for the eigenvalues of an arbitrarily shaped metallic cavity using node-based and edge-based vector finite elements. The computed data are then compared with analytical results for empty and partially filled cavities. A comparison between the storage intensity and computational accuracy for edge-based rectangular bricks and tetrahedra is also presented. Finally, we compute the eigenvalues of a metallic cavity with a ridge along one of its faces.

2 Formulation

2.1 Derivation of finite element equations

Consider a three dimensional inhomogeneous body occupying the volume V. To discretize the electric field **E** inside this volume, we subdivide it into a number of small tetrahedra or rectangular bricks, each occupying the volume $V_e(e = 1, 2, \cdots, M)$, where M is the total number of elements. Within each element, the electric field satisfies the vector wave equation

$$\nabla \times \frac{1}{\mu_r} \nabla \times \mathbf{E} - k_o^2 \epsilon_r \mathbf{E} = \mathbf{0} \tag{1}$$

in which μ_r is the permeability of the medium, ϵ_r is the medium permittivity and k_o is the free space wave number. For a numerical solution of (1), we expand the electric field within the eth volume element as

$$\mathbf{E} = \sum_{j=1}^{m} E_j^e \mathbf{W}_j^e \tag{2}$$

where $\mathbf{W}_j^e$ are edge-based vector basis functions, E_j^e denote the expansion coefficients of the basis, m represents the number of edges in the element and the superscript stands for the element number. On substituting (2) into (1) and upon applying Galerkin's technique, we obtain

$$\sum_{j=1}^{m} E_j^e \int_{V_e} \mathbf{W}_i^e \cdot \left(\nabla \times \frac{1}{\mu_r} \nabla \times \mathbf{W}_j^e - k_o^2 \epsilon_r \mathbf{W}_j^e \right) dv \;\; = \;\; 0$$

Further, upon making use of basic vector identities and the divergence theorem, we obtain the weak form of Maxwell's equation

$$\sum_{j=1}^{m} E_j^e \int_{V_e} \left[\frac{1}{\mu_r} (\nabla \times \mathbf{W}_i^e) \,.\, (\nabla \times \mathbf{W}_j^e) - k_o^2 \epsilon_r \mathbf{W}_i^e . \mathbf{W}_j^e \right] dv = \jmath k_o Z_o \oint_{S_e} \mathbf{W}_i^e . (\mathbf{n} \times \mathbf{H}) ds \tag{3}$$

where $\mathbf{n} \times \mathbf{H}$ is the tangential magnetic field on the exterior dielectric surface, S_e denotes the surface enclosing V_e and Z_o is the free-space intrinsic impedance. Equation (3) can be conveniently written in matrix form as

$$[A^e] \{E^e\} \;\; = \;\; k_o^2 [B^e] \{E^e\} + \{C^e\} \tag{4}$$

where

$$A_{ij}^e \;\; = \;\; \int_{V_e} \frac{1}{\mu_r} (\nabla \times \mathbf{W}_i^e) \,.\, (\nabla \times \mathbf{W}_j^e) \tag{5}$$

$$B_i^e \;\; = \;\; \int_{V_e} \epsilon_r \mathbf{W}_i^e . \mathbf{W}_j^e dv \tag{6}$$

$$C_i^e \;\; = \;\; \jmath k_o Z_o \oint_{S_e} \mathbf{W}_i^e . (\mathbf{n} \times \mathbf{H}) ds \tag{7}$$

On assembling the equations from all the elements making up the geometry, we obtain the system

$$\sum_{e=1}^{M}[A^e]\{E^e\} = k_o^2\sum_{e=1}^{M}[B^e]\{E^e\} + \sum_{e=1}^{M}\{C^e\} \quad (8)$$

Due to the continuity of tangential **H** at the interface between two dielectrics, an element face lying inside the body does not contribute to the last term of (8) in the final assembly of the element equations. As a result, the last term of (8) reduces to a column vector containing the surface integral of the tangential magnetic field only over the exterior surface of the body. In this paper, we are interested in a perfectly conducting surface surrounding the volume V and in this case, the surface integral vanishes altogether. Therefore, this system can be more compactly written as

$$[A]\{E\} = \lambda[B]\{E\} \quad (9)$$

where $[A]$ and $[B]$ are $N \times N$ symmetric, sparse matrices with N being the total number of edges resulting from the subdivision of the body, $\{E\}$ is a $N \times 1$ column vector denoting the edge fields and $\lambda = k_o^2$ gives the eigenvalues of the system. Solving (9), we obtain the resonant wavenumber k_o and the resonant field distribution $\{E\}$ as well.

In the case of the scattering problem, (3) can be conveniently written in matrix form as

$$\left[A'^e\right]\{E^e\} = \{C^e\} \quad (10)$$

where

$$A'^e_{ij} = \int_{V_e}\left(\frac{1}{\mu_r}(\nabla\times\mathbf{W}_i^e).(\nabla\times\mathbf{W}_j^e) - k_o^2\epsilon_r\mathbf{W}_i^e.\mathbf{W}_j^e\right)dv \quad (11)$$

$$C_i^e = \jmath k_o Z_o \oint_{S_e}\mathbf{W}_i^e.(\mathbf{n}\times\mathbf{H})ds \quad (12)$$

Summing over all M elements in the geometry, we obtain a system of equations whose solution yields the field components over the entire body.

$$\sum_{e=1}^{M}\left[A'^e\right]\{E^e\} = \sum_{e=1}^{M}\{C^e\}$$

which gives

$$[A']\{E\} = \{C\} \quad (13)$$

where $\{E\}$ are defined as in (9), $[A']$ is a $N \times N$ matrix and $\{C\}$ is a column vector of size N related to the tangential magnetic field over the exterior surface of the body. On separating the on-surface edges from the edges inside the body for ease of representation, we obtain

$$\begin{aligned}[A'_{ss}]\{E_s\} + [A'_{si}]\{E_i\} &= \{C_s\}\\ [A'_{is}]\{E_s\} + [A'_{ii}]\{E_i\} &= 0\\ \{C_s\} &= [B'_{ss}]\{H_s\}\end{aligned} \quad (14)$$

where the subscript s denotes the edges on the surface and i represents the edges inside the body. It is thus readily seen that (14) relates the electric field inside and on the surface of the body to the on-surface tangential magnetic field. However, (14) contains two unknown vectors $\{E\}$ and $\{H\}$ and the system can, therefore, be solved only when we relate the electric field with the magnetic field. This is done by introducing an absorbing boundary condition at the outer boundary of the mesh as discussed in section 2.3.

2.2 Basis functions

The vector edge basis functions for rectangular bricks are outlined in detail in [9]. The basis functions for tetrahedral elements merit detailed mention since they are more involved.

Vector fields within tetrahedral domains in three dimensional space can be conveniently represented by expansion functions that are linear in the spatial variables and have either zero divergence or zero curl. The basis functions defined in [8] are associated with the six edges of the tetrahedron and have zero divergence and constant curl. Assuming the four nodes and the six edges of a tetrahedron are numbered according to Table 1, the vector basis functions associated with the $(7-i)$th edge of the tetrahedron are defined as

$$\mathbf{W}_{7-i} = \begin{cases} \mathbf{f}_{7-i} + \mathbf{g}_{7-i} \times \mathbf{r}, & \mathbf{r} \text{ in the tetrahedron} \\ 0, & \text{otherwise} \end{cases} \tag{15}$$

with

$$\mathbf{f}_{7-i} = \frac{b_{7-i}}{6V} \mathbf{r}_{i_1} \times \mathbf{r}_{i_2} \tag{16}$$

$$\mathbf{g}_{7-i} = \frac{b_i b_{7-i} \mathbf{e}_i}{6V} \tag{17}$$

where $i = 1, 2, \cdots, 6$, V is the volume of the tetrahedral element, $\mathbf{e}_i = (\mathbf{r}_{i_2} - \mathbf{r}_{i_1})/b_i$ is the unit vector of the ith edge and $b_i = |\mathbf{r}_{i_2} - \mathbf{r}_{i_1}|$ is the length of the ith edge. All distances are measured with respect to the origin.

Since there are two numbering systems, local and global, a unique global edge direction is defined (e.g., always pointing from the smaller node number to the larger node number)[10] to ensure the continuity of $\mathbf{n} \times \mathbf{E}$ across all edges. This implies that (15) should be multiplied by (-1) if the local edge vector (as defined in Table 1) does not have the same direction as the global edge direction, i.e. the local vector points from the larger node number to the smaller node number. Finally, since $\nabla \cdot \mathbf{W}_i = 0$ the electric field obtained through (2) exactly satisfies the divergence equation within the element, i.e. $\nabla \cdot \mathbf{E} = 0$. Therefore, the finite element solution is free from contamination of spurious solutions.

2.3 Mesh termination

Differential equation methods, such as finite elements, can only solve boundary value problems. Since electromagnetic problems are open boundary-

infinite domain types, a means to truncate the solution domain to lie within a finite boundary must be found. On this boundary, a condition is enforced thus ensuring that the fields will obey the Sommerfeld radiation condition at distances asymptotically far from the object. These absorbing boundary conditions (ABCs) have a significant advantage over the global methods of solving unbounded problems using finite elements in that they are local in nature. Due to this, the sparse matrix structure of the finite element formulation is retained. One disadvantage, however, is that ABCs are approximate and do not model the exterior field exactly.

The objective of absorbing boundary conditions is to truncate the finite element mesh with boundary conditions that cause minimum reflections of an outgoing wave. These ABCs should provide small, acceptable errors while minimising the distance from the object of interest to the outer boundary. This minimal distance is required to reduce the number of unknowns in the problem for computational efficiency.

There are two ways to approach this problem. The first approach is to employ a three dimensional vector boundary condition for terminating the finite element mesh of the body described in section 2.1. The second method involves placing an artificial conducting boundary (electric or magnetic) to truncate the infinite region and then coating the inner surface of the boundary with a layer or several layers of fictitious dielectric whose thickness and constitutive parameters are chosen to minimise the non-physical reflections of the scattered field over a wide range of incidence angles[11].

2.3.1 Vector ABC

We begin with the Wilcox representation[12] of the electric field which has an expansion

$$\mathbf{E}(\mathbf{r}) = \frac{e^{-jkr}}{r} \sum_{n=0}^{\infty} \frac{\mathbf{A}_n(\theta, \phi)}{r^n} \tag{18}$$

From (18), we get

$$\nabla \times \mathbf{E} = \left\{ jk\hat{r} \times \frac{1 + D_1}{r} \right\} \mathbf{E} - \frac{e^{-jkr}}{r^2} \sum_{n=1}^{\infty} \frac{n\mathbf{A}_{nt}}{r^n} \tag{19}$$

where $\mathbf{A}_{nt} = \hat{r} \times \mathbf{A}_n$ is the transverse component of $\mathbf{A}_n$ and, for a vector $\mathbf{F}$, $D_1\mathbf{F}$ is given by

$$\begin{aligned} D_1\mathbf{F} &= \frac{1}{sin\theta} \left[\frac{\partial}{\partial\theta}(sin\theta F^{\phi}) - \frac{\partial F^{\theta}}{\partial\phi} \right] \hat{r} \\ &+ \frac{1}{sin\theta} \left[\frac{\partial F^{r}}{\partial\theta} - sin\theta F^{\phi} \right] \hat{\theta} + \left[F^{\theta} - \frac{\partial F^{r}}{\partial\theta} \right] \hat{\phi} \end{aligned} \tag{20}$$

Using the recursion relation

$$-2jkn\mathbf{A}_{nt} = n(n-1)\mathbf{A}_{n-1,t} + D_4\mathbf{A}_{n-1}$$

where

$$
\begin{aligned}
D_4\mathbf{A}_n &= \left(D\ A_n^\theta + D_\theta\ \mathbf{A}_n\right)\hat{\theta} + \left(D\ A_n^\phi + D_\phi\ \mathbf{A}_n\right)\hat{\phi} \\
D_\theta\ \mathbf{A}_n &= 2\frac{\partial A_n^r}{\partial\theta} - \frac{1}{sin^2\theta}A_n^\theta - \frac{2cos\theta}{sin^2\theta}\frac{\partial A_n^\phi}{\partial\theta} \\
D_\phi\ \mathbf{A}_n &= \frac{2}{sin\theta}\frac{\partial A_n^r}{\partial\phi} - \frac{1}{sin^2\theta}A_n^\phi + \frac{2cos\theta}{sin^2\theta}\frac{\partial A_n^\theta}{\partial\phi}
\end{aligned}
$$

and D is Beltrami's operator[10], we can derive the representation correct to r^{-4}. Applying the recursion relation in (19) yields the desired relationship for the vector ABC:

$$\nabla \times \mathbf{E} = \alpha(r)\mathbf{E} + \beta(r)D_4\mathbf{E} \tag{21}$$

where

$$\alpha(r) = jk\left\{\frac{D_1}{jkr} - \left(1 + \frac{1}{jkr}\right)\hat{r}\times\right\} \tag{22}$$

$$\beta(r) = \frac{1}{2jkr^2}\frac{1}{(1 + 1/jkr)} \tag{23}$$

The ABC formulated above was derived for spherical boundaries and hence would be storage intensive and numerically inefficient when used to terminate the mesh of long and thin geometries. It would be highly desirable to choose an outer boundary that conforms to the shape of the object. It is the authors' intention to extend the boundary condition discussed above to conformal boundaries. Another way to introduce a conformal outer boundary is to use the fictitious absorber model discussed below.

2.3.2 Fictitious absorber model

In this section, our intention is to develop a model which can absorb the reflections of the scattered field from the artificial outer boundary over as wide a range of incidence angles as possible. The reflection coefficient on applying the appropriate boundary condition can be symbolically written as

$$R = f(t_1, \epsilon_{r1}, \mu_{r1}; t_2, \epsilon_{r2}, \mu_{r2}; \cdots; t_n, \epsilon_{rn}, \mu_{rn}; \lambda, \theta) \tag{24}$$

for an n-layer absorber. Here, t_i, ϵ_{ri} and μ_{ri} denote the thickness, relative permittivity and relative permeability of the ith dielectric layer. Using a multi-dimensional minimization algorithm such as that based on the downhill simplex method, we can find a set of $t_i, \epsilon_{ri}, \mu_{ri}$ that minimises the magnitude of the reflection coefficient over a wide range of incidence angles. Further, the outer boundary of the mesh can be made conformal to the shape of the scatterer reducing the number of unknowns compared to the earlier approach which works solely for spherical outer boundaries.

2.4 Solution of the finite element equations

2.4.1 Vector ABC

An inspection of (13) reveals that for an inhomogeneous body, there is no a priori information about the tangential magnetic field over the exterior surface of the body. Relation (13) therefore contains two unknown vectors, $\{E\}$ and $\{C\}$, and thus another condition is required involving the two variables to permit an evaluation of the fields inside and on the surface of the body. This condition relating the tangential electric field to the tangential magnetic field on the surface is provided by (21). Since the ABC in (21) refers to the scattered field, we can rewrite it as

$$\begin{aligned} \nabla \times \mathbf{E}_s^s &= \alpha(r)\mathbf{E}_s^s + \beta(r)D_4\mathbf{E}_s^s \\ \mathbf{H}_s^s &= \frac{j}{\omega\mu}\left[\alpha(r)\mathbf{E}_s^s + \beta(r)D_4\mathbf{E}_s^s\right] \\ &= \mathcal{K}\mathbf{E}_s^s \end{aligned} \tag{25}$$

where $\mathcal{K} = \frac{j}{\omega\mu}\left[\alpha(r) + \beta(r)D_4\right]$ and the subscript s denotes the field on the surface and the superscript s represents the scattered field. As the total field is a sum of the incident field and the scattered field, therefore from (25), we obtain

$$\mathbf{H}_s - \mathbf{H}_s^{inc} = \mathcal{K}\left(\mathbf{E}_s - \mathbf{E}_s^{inc}\right) \tag{26}$$

Since $\{H_s\} = [B_{ss}]^{-1}\{C_s\}$ from (14), we obtain on substituting (26) into (14)

$$\begin{aligned} \left([A'_{ss}] - [B_{ss}]\mathcal{K}\right)\{E_s\} + [A'_{si}]\{E_i\} &= [B_{ss}]\left(\left\{H_s^{inc}\right\} - \mathcal{K}\left\{E_s^{inc}\right\}\right) \\ [A'_{is}]\{E_s\} + [A'_{ii}]\{E_i\} &= 0 \end{aligned} \tag{27}$$

The above equation can thus be solved for the unknown electric fields both inside and on the surface of the body.

2.4.2 Fictitious absorber model

Since an artificial outer boundary is placed in this case, the column vector $\{C\}$ in (13) is a function of the incident field and the integration is carried out over the outer and inner surfaces. Therefore, we can rewrite (14) as

$$\begin{aligned} [A''_{ss}]\{E_s^s\} + [A''_{si}]\{E_i^s\} &= \left\{C_s^{inc}\right\} \\ [A''_{is}]\{E_s^s\} + [A''_{ii}]\{E_i^s\} &= 0 \end{aligned} \tag{28}$$

where the superscript *inc* stands for the incident field, the superscript s denotes the scattered field, the subscripts are as defined in (14) and $[A'']$ contains contributions from the surface integral as well as $[A']$.

3 Results

In Table 2, the eigenvalues for a 1cm×.5cm×.75cm rectangular cavity are calculated using nodal basis functions. This particular derivation is outlined in [13] and the existence of 'spurious' modes is clearly observed when the zero divergence condition is not rigidly enforced ($s = 0$). For non-zero values of s, 'spurious' modes do not appear. However, arbitrary values of s lead to inaccuracies and, therefore, the most suitable value needs to be determined by trial and error.

Using edge-based rectangular bricks results in better accuracy as observed in Table 3. However, on comparing bricks with tetrahedra, it is seen from Table 4 that the tetrahedral edge element is the best in terms of accuracy for the same storage requirement. The maximum edge length for the rectangular brick elements was .15cm whereas that for the tetrahedral elements was .2cm.

In Tables 4-7, we compare the eigenvalues computed using edge-based tetrahedral finite elements with their analytical counterparts. The finite element mesh was generated using SDRC I-DEAS, a commercial pre-processing package and it is seen that the numerical results are in excellent agreement with the exact values for both homogeneous and inhomogeneous cavities. In Table 4, the edge-based approach predicts the first six distinct non-trivial eigenvalues with less than 4 per cent error for an empty rectangular cavity. The exact eigenvalues of the half-filled cavity as described in Table 5 are computed by solving the transcendental equation obtained by matching the tangential electric and magnetic fields at the air-dielectric interface. As seen, these results agree with the results of the finite element code to within 1 per cent. Table 6 compares the analytical and computed eigenvalues of a cylindrical cavity having a base radius of 0.5cm and a height of 0.5cm. The eigenvalues are again seen to be quite accurate and the accuracy is expected to increase further with higher mesh density. Similar comparisons are given in Table 7 of a sphere of 1cm radius. Finally, Table 8 presents the eigenvalues of the geometry shown in Fig. 1. This is basically a closed metallic cavity with a ridge along one of its faces. We note again that it is difficult to treat such a geometry having sharp edges and corners using node-based finite elements. The storage requirement for the geometries discussed above can be further reduced by taking symmetry into account. Since we are interested in developing a code for arbitrary geometries, boundary conditions taking advantage of symmetry have not been introduced.

It is noted that as the degeneracy of the eigenvalues increases, the matrix becomes increasingly ill-conditioned and the numerical solution correspondingly less accurate[14]. This is clearly observed from the data in Table 7 for the case of a perfectly conducting hollow spherical cavity. Since the second lowest TM mode has five-fold degeneracy, the computational error is seen to be the greatest. However, for the partially filled rectangular cavity, the absence of degenerate modes gives results which are accurate to within 1 per cent of the exact eigensolutions.

4 Conclusions and future work

It has been shown that the resonant frequencies of an arbitrarily shaped inhomogeneously filled metallic resonator can be computed very accurately using edge-based tetrahedral elements. The nodal method is not very reliable and is dependent on the value of the s parameter which is a measure of how strongly one wants to enforce the divergence-free condition. Edge-based rectangular bricks do not provide as good an accuracy as edge-based tetrahedral elements and is limited to a small class of geometries. It has, therefore, been established that it is best to use edge-based tetrahedral elements for solving three-dimensional problems.

In the future, we intend to work on three dimensional scattering problems using edge-based tetrahedra and employing the mesh termination techniques discussed in section 2.3. We also plan to focus on the derivation of new absorbing boundary conditions conformal to the shape of the scatterer.

5 Appendix

5.1 Derivation of matrix elements

The derivation of the matrix elements in (9) amounts to evaluating the integrals in (5) and (6). Therefore, from (5), we have

$$\int_{V_e} \frac{1}{\mu_r}(\nabla \times \mathbf{W}_i^e) \,.\, (\nabla \times \mathbf{W}_j^e) \;=\; \frac{4}{\mu_r}\mathbf{g}_i.\mathbf{g}_j V_e \tag{29}$$

since $\nabla \times \mathbf{W}_i^e = 2\mathbf{g}_i$. The evaluation of the integral in (6) is more cumbersome. Substituting into (6) the basis functions defined in (16) and (17), we obtain

$$\begin{aligned} \epsilon_r \int_{V_e} \mathbf{W}_i^e.\mathbf{W}_j^e dv &= \epsilon_r \int_{V_e} \{(\mathbf{f}_i.\mathbf{f}_j) + (\mathbf{r}.\mathbf{D}) + (\mathbf{g}_i \times \mathbf{r}).(\mathbf{g}_j \times \mathbf{r})\}\, dv \qquad (30) \\ &= \epsilon_r\,(I_1 + I_2 + I_3) \end{aligned}$$

where

$$\mathbf{D} \;=\; (\mathbf{f}_i \times \mathbf{g}_j) + (\mathbf{f}_j \times \mathbf{g}_i)$$

and

$$I_1 \;=\; \int_{V_e} \mathbf{f}_i.\mathbf{f}_j dv \tag{31}$$

$$I_2 \;=\; \int_{V_e} \mathbf{r}.\mathbf{D} dv \tag{32}$$

$$I_3 \;=\; \int_{V_e} (\mathbf{g}_i \times \mathbf{r}).(\mathbf{g}_j \times \mathbf{r}) dv \tag{33}$$

Since $\mathbf{f}$ is a constant vector, I_1 reduces to

$$I_1 = \mathbf{f}_i.\mathbf{f}_j\, V_e \tag{34}$$

Since

$$x = \sum_{i=1}^{4} L_i x_i$$

$$y = \sum_{i=1}^{4} L_i y_i$$

$$z = \sum_{i=1}^{4} L_i z_i$$

where L_i are the shape functions for the tetrahedral element and $x_i, y_i, z_i (i = 1, \cdots, 4)$ denote the x, y and z co-ordinates of the vertices of the tetrahedral element. Using the standard formula for volume integration within a tetrahedral element and simplifying, we have

$$I_2 = \frac{V_e}{4}\left[D_x \sum_{i=1}^{4} x_i + D_y \sum_{i=1}^{4} y_i + D_z \sum_{i=1}^{4} z_i\right] \tag{35}$$

where D_m is the mth component of $\mathbf{D}$. The evaluation of I_3 can be simplified by the use of basic vector identities. Therefore,

$$\begin{aligned} I_3 &= \mathbf{g}_i.\mathbf{g}_j \int_{V_e} |\mathbf{r}|^2\, dv - \int_{V_e} (\mathbf{g}_i.\mathbf{r})\,(\mathbf{g}_j.\mathbf{r})\, dv \\ &= (g_{iy}g_{jy} + g_{iz}g_{jz}) \int_{V_e} x^2 dv + (g_{ix}g_{jx} + g_{iz}g_{jz}) \int_{V_e} y^2 dv + (g_{ix}g_{jx} + g_{iy}g_{jy}) \int_{V_e} z^2 dv \\ &\quad - (g_{ix}g_{jy} + g_{jx}g_{iy}) \int_{V_e} xy dv - (g_{ix}g_{jz} + g_{jx}g_{iz}) \int_{V_e} zx dv - (g_{iy}g_{jz} + g_{jy}g_{iz}) \int_{V_e} yz dv \end{aligned} \tag{36}$$

where g_{im} represents the mth component of the vector $\mathbf{g}_i$. Each of the integrals in (36) can be easily evaluated analytically and the result expressed in the following general form:

$$\int_{V_e} a_l a_m dv = \frac{V_e}{20}\left[\sum_{i=1}^{4} a_{li}a_{mi} + \sum_{i=1}^{4} a_{li} \sum_{i=1}^{4} a_{mi}\right] \tag{37}$$

where $l, m = 1, \cdots, 3$ and a_1 represents the variable x, a_2 stands for the variable y and a_3 denotes the variable z.

6 References

1. Z.J. Cendes and P. Silvester, "Numerical solution of dielectric loaded waveguides: I - Finite element analysis", *IEEE Trans. Microwave Theory Tech.*, vol. 18, pp.1124-31, 1970.

2. B.M.A. Rahman and J.B. Davies, "Penalty function improvement of

waveguide solution by finite elements", *IEEE Trans. Microwave Theory Tech.*, vol. 32, pp.922-8, Aug. 1984.
3. J.P. Webb, "The finite-element method for finding modes of dielectric-loaded cavities", *IEEE Trans. Microwave Theory Tech.*, vol. 33, no.7, pp.635-9, July 1985.
4. J.P. Webb, "Finite element analysis of dispersion in waveguides with sharp metal edges", *IEEE Trans. Microwave Theory Tech.*, vol. 36, no.12, pp.1819-24, Dec. 1988.
5. X. Yuan, D.R. Lynch and K. Paulsen, "Importance of normal field continuity in inhomogeneous scattering calculations", *IEEE Trans. Microwave Theory Tech.*, vol. 39, no.4, pp.638-42, April 1991.
6. A. Bossavit, "A rationale for 'edge-elements' in 3-D fields computations", *IEEE Trans. Magn.*, vol. 24, no.1, pp.74-9, Jan. 1988.
7. A. Bossavit, "Solving Maxwell equations in a closed cavity, and the question of 'spurious' modes", *IEEE Trans. Magn.* , vol. 26, no.2, pp.702-5, March 1990.
8. M.L. Barton and Z.J. Cendes, "New vector finite elements for three-dimensional magnetic field computation", *J. Appl. Phys.*, vol. 61, no.8, pp.3919-21, April 1987.
9. J.M. Jin and J.L. Volakis, "Electromagnetic scattering by and transmission through a three-dimensional slot in a thick conducting plane", *IEEE Trans. Antennas Propagat.*, vol. 39, no.4, pp. 543-50, April 1991.
10. X. Yuan, "Three-dimensional electromagnetic scattering from inhomogeneous objects by the hybrid moment and finite element method", *IEEE Trans. Microwave Theory Tech.*, vol. 38, no.8, pp. 1053-9, Aug. 1990.
11. J.M. Jin, J.L. Volakis and V.V. Liepa, "An engineer's approach for terminating finite element meshes in scattering analysis", to appear.
12. C.H. Wilcox, "An expansion theorem for electromagnetic fields", *Comm. Pure Appl. Math.*, vol. 9, pp. 115-134, May 1956.
13. J.M. Jin and J.L. Volakis, "A finite element-boundary integral formulation for scattering by three-dimensional cavity-backed apertures", *IEEE Trans. Antennas Propagat.*, vol. 39, no.1, pp. 97-104, Jan. 1991.
14. G.H. Golub and C.F. Van Loan, *Matrix Computations*, pp. 202-4, The Johns Hopkins University Press, 1985.

Edge no.	i_1	i_2
1	1	2
2	1	3
3	1	4
4	2	3
5	4	2
6	3	4

Table 1: Tetrahedron edge definition

Mode no.	Exact	s=0	s=.5	s=1	s=5
spurious		2.010			
spurious		3.312			
spurious		4.443			
spurious		4.928			
TE101	5.236	5.318	5.427	5.447	5.571
spurious		5.759			
spurious		6.255			
spurious		6.804			
TM 110	7.025	7.067	6.574	7.543	7.940
spurious		7.305			
TE 011	7.531	7.389	7.476	7.635	8.014
TE 201	7.531	7.640	7.659	8.072	8.348

Table 2: Eigenvalues for an empty 1cm ×0.5cm ×0.75cm rectangular cavity computed using nodal basis (230 unknowns)

Mode no.	Analytical	Computed	Error(%)
TE 101	5.236	5.307	-1.36
TM 110	7.025	7.182	-2.23
TE 011	7.531	7.725	-2.58
TE 201	7.531	7.767	-3.13
TM 111	8.1789	8.350	-2.09
TE 111	8.1789	8.350	-2.09
TM 210	8.886	9.151	-2.98
TE 102	8.947	9.428	-5.38

Table 3: Eigenvalues for an empty 1cm ×0.5cm ×0.75cm rectangular cavity using edge-based rectangular bricks (270 unknowns)

Mode no.	Analytical	Computed	Error(%)
TE 101	5.236	5.213	.44
TM 110	7.025	6.977	.70
TE 011	7.531	7.474	1.00
TE 201	7.531	7.573	-.56
TM 111	8.1789	7.991	2.29
TE 111	8.1789	8.122	.70
TM 210	8.886	8.572	3.53
TE 102	8.947	8.795	1.70

Table 4: Eigenvalues for an empty 1cm $\times$0.5cm $\times$0.75cm rectangular cavity (260 unknowns)

Mode no.	Analytical	Computed	Error(%)
TEz 101	3.538	3.534	.11
TEz 201	5.445	5.440	.10
TEz 102	5.935	5.916	.32
TEz 301	7.503	7.501	.04
TEz 202	7.633	7.560	.97
TEz 103	8.096	8.056	.50

Table 5: Eigenvalues for a half-filled 1cm $\times$0.1cm $\times$1cm rectangular cavity ($\epsilon_r = 2$) (192 unknowns)

Mode no.	Analytical	Computed	Error(%)
TM 010	4.810	4.809	.02
TE 111	7.283	7.202	1.1
		7.288	-.07
TM 110	7.650	7.633	.22
		7.724	-.97
TM 011	7.840	7.940	-1.28
TE 211	8.658	8.697	-.45
		8.865	-2.39

Table 6: Eigenvalues for an empty cylindrical cavity of base radius 0.5cm and height 0.5 cm (380 unknowns)

Mode no.	Analytical	Computed	Error(%)
TM 010	2.744	2.799	-2.04
TM 111,even		2.802	-2.11
TM 111,odd		2.811	-2.44
TM 021	3.870	3.948	-2.02
TM 121,even		3.986	-2.99
TM 121,odd		3.994	-3.20
TM 221,even		4.038	-4.34
TM 221,odd		4.048	-4.59
TE 011	4.493	4.433	1.33
TE 111,even		4.472	.47
TE 111,odd		4.549	-1.25

Table 7: Eigenvalues for an empty spherical cavity of radius 1cm (300 unknowns)

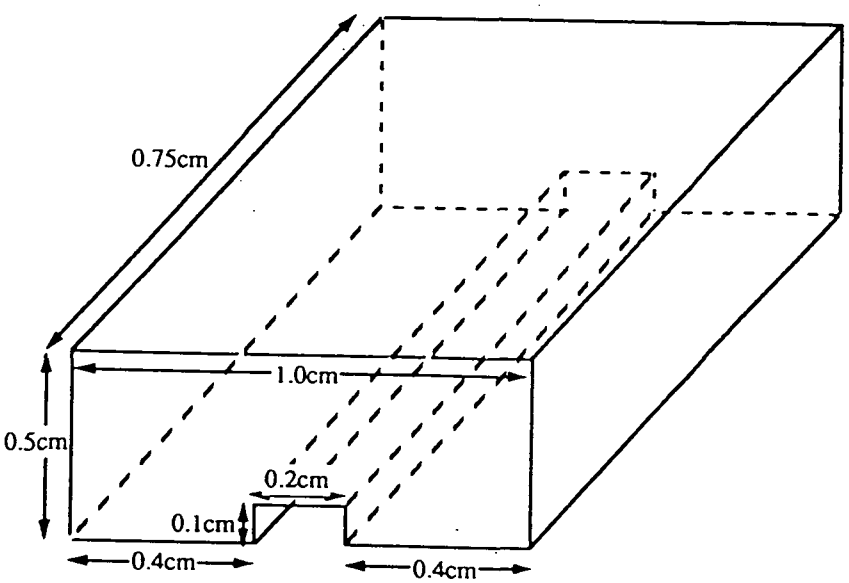

Figure 1: A ridged cavity

1	4.941
2	7.284
3	7.691
4	7.855
5	8.016
6	8.593
7	8.906
8	9.163
9	9.679
10	9.837

Table 8: Ten lowest non-trivial eigenvalues of Fig. 1 (267 unknowns)

The following is a listing of the computer programs generating the node-based and edge-based elements, interfacing with SDRC IDEAS and computing the eigenvalues of an inhomogeneously filled, arbitrarily shaped cavity.

```
================================================================================
Processes a universal file obtained from IDEAS and converts the nodal info
to edge info needed for constructing an edge-based three-dimensional finite
element mesh using tetrahedral elements.
Stores the node numbers and respective nodal coordinates in 'noddy'
Stores the element numbers and corresponding nodes in 'elno'
Processes 'noddy' and 'elno' and stores the edge nos. and nodal connections
in 'glob' and edge nos. with corresponding edge vector in 'edgy'
Note: Edge overlaps are taken care of.
Storage limit: 800 nodes, 3000 elements, 4300 edges
================================================================================
    program unv_file_processor
    character string*80, yastrn*40, unv*20
    integer n1(3000,4),tab(4300,2),nn(100),tr(3000),nun(3000),
 1          bc(3000,3),edgv(18000,3),mat(3000)
    real x(800),y(800),z(800)
    common /bank/n1,x,y,z,ne,mat
    common /dbase/edgv
    common /local/ncount
    write(6,*)'1) Cavity problem 2) Scattering problem'
    read(5,*)ivar
    write(6,*)'Name of universal file ?'
    read(5,'(A)')unv
    open(1,file=unv)
    open(2,file='enoddy')
    open(3,file='elno')
    open(4,file='eglob')
    open(7,file='edgy')
    open(8,file='esurfed')
.Processing universal file for info on various parameters
    read(1,'(A)')string
    l=0
    if (string(4:6).eq.'151') then
      write(6,*)'Encountered header'
      go to 15
    elseif (string(4:6).eq.'747') then
      read(1,'(A)')string
      if (string(5:6).ne.'-1') then
        if (string(9:9).ne.' ') then
          write(4,*)string(2:13)
          do ik=1,4
            read(1,'(A)')string
          enddo
        endif
        go to 2
      endif
.Processing nodal info; data stored in 'noddy'
    elseif (string(4:6).eq.' 15') then
      read(1,'(A)')string
      if (string(5:6).ne.'-1') then
        write(2,*)string(7:10),' ',string(41:53),' ',string(54:66),' '
 1               ,string(67:79)
        l=l+1
        go to 5
      endif
      l1=l
      write(6,*)'There are ',l,' nodes.'
      write(4,*)l
.Processing element info; data stored in 'elno'
```

```
    elseif (string(4:6).eq.' 71') then
      read(1,'(A)')string
      if (string(5:6).ne.'-1') then
        read(1,'(A)')yastrn
        write(3,*)string(7:10),' ',yastrn(7:10),' ',yastrn(17:20),
 1      ' ',yastrn(27:30),' ',yastrn(37:40),'    ',string(49:50)
        l=l+1
        go to 10
      else
        write(6,*)'There are ',l,' elements.'
        write(4,*)l
        l2=l
      endif
    elseif (string(4:6).eq.'752') then
      ltmp=0
      nn(1)=0
      read(1,'(A)')string
      if (string(19:20).eq.' 0') then
        ltmp=ltmp+1
        read(string(57:60),'(I4)')numb
        nn(ltmp+1)=numb+nn(ltmp)
      elseif (string(5:6).eq.'-1') then
        write(6,*)'There are ',ltmp,' surfaces.'
        go to 20
      elseif (string(1:1).eq.' ') then
        do i=1,4
          if (string((20*i-3):20*i).ne.'    ') then
            l=l+1
            read(string((20*i-3):20*i),'(I4)')tr(l)
          endif
        enddo
      endif
      go to 30
    endif
    go to 15
    close(1)
.Data from 'noddy' and 'elno' stored in
 nl    - table of nodes making up the corresponding element
 x,y,z - nodal coordinate table
    rewind 2
    rewind 3
    do i=1,l1
      read(2,*)na,x(i),y(i),z(i)
    enddo
    do i=1,l2
      read(3,*)nk,nl(i,1),nl(i,2),nl(i,3),nl(i,4),mat(i)
    enddo
.Close 'noddy' and 'elno' for good
    close(2)
    close(3)
    ncount=0
    write(6,*)'Be patient #*!?/#@*!!!'
    do ne=1,l2
.Store edge info in an integer array 'tab' after checking for
 overlap. The subroutine 'compare' is the heart of the program.
      call compare(tab)
.Commenting out the following statement causes a speedup of 250%
      write(6,*)'Processed element no. ',ne,' :edge count= ',ncount
    enddo
```

```
      write(6,*)'Edge count = ',ncount
      rewind(4)
      write(6,*)'No. of dielectric layers?'
      read(5,*)itemp
      read(4,*)(xtmp,i=1,itemp)
      read(4,*)(nj,i=1,2)
      do i=1,(6*l2)
        read(4,*)nk,edgv(i,1),edgv(i,2),edgv(i,3),nj
      enddo
      close(4)
      close(7)
      nsurf=0
      if (ivar.eq.1) then
        do i=1,ltmp
          write(6,*)'Processing surface ',i,' for on-surface edges.'
          do j=nn(i)+1,nn(i+1)
            do k=nn(i)+1,nn(i+1)
              do l=1,ncount
                if (tab(l,1).ne.0) then
                  if ((tr(j).eq.tab(l,1)).and.(tr(k).eq.tab(l,2))) then
                    write(8,*)l
                    tab(l,1)=0
                    nsurf=nsurf+1
                    go to 40
                  endif
                endif
              enddo
            enddo
          enddo
        enddo
        write(6,*)'Total no. of on-surface edges= ',nsurf
        write(6,*)'Fem matrix to be solved is of order ',(ncount-nsurf)
      else
        ncnt=0
        do ii=1,ltmp
          do ik=1,l2
            nun(ik)=0
          enddo
          do i=nn(ii)+1,nn(ii+1)
            do j=1,l2
              do k=1,4
                if (tr(i).eq.nl(j,k)) then
                  nun(j)=nun(j)+1
                  bc(j,nun(j))=nl(j,k)
                  if (nun(j).eq.3) then
                    call edge(j,bc(j,1),bc(j,2),bc(j,3))
                    ncnt=ncnt+1
                    go to 80
                  endif
                endif
              enddo
            enddo
          enddo
        enddo
        write(6,*)'There are ',ncnt,' surface elements.'
      endif
      close(8)
      stop
      end
```

```
C=======================================================================
C 'compare' checks for all possible edges avoiding overlap and stores the
C  info in an array 'tab'. The file 'edgy' contains the element no.
C  and the six edge vectors corresponding to that element.
C=======================================================================
      subroutine compare(tab)
      integer n1(3000,4),mat(3000)
      real x(800),y(800),z(800)
      integer nt(6,2),tab(4300,2)
      common /bank/n1,x,y,z,ne,mat
      common /local/ncount
      nt(1,1)=n1(ne,1)
      nt(1,2)=n1(ne,2)
      nt(2,1)=n1(ne,1)
      nt(2,2)=n1(ne,3)
      nt(3,1)=n1(ne,1)
      nt(3,2)=n1(ne,4)
      nt(4,1)=n1(ne,2)
      nt(4,2)=n1(ne,3)
      nt(5,1)=n1(ne,2)
      nt(5,2)=n1(ne,4)
      nt(6,1)=n1(ne,3)
      nt(6,2)=n1(ne,4)
C.Arranging node couplets in ascending order; a personal choice
      do i=1,6
        if (nt(i,1).gt.nt(i,2))then
          ntmp=nt(i,1)
          nt(i,1)=nt(i,2)
          nt(i,2)=ntmp
        endif
      enddo
C.A brute force search for overlapping edges
      l=ncount
      do ii=1,6
        if (ne.eq.1) go to 32
        do i=1,ncount
          if ((tab(i,1).eq.nt(ii,1)).and.(tab(i,2).eq.nt(ii,2))) then
            write(4,*)ne,i,nt(ii,1),nt(ii,2),mat(ne)
            go to 35
          endif
        enddo
        l=l+1
        tab(l,1)=nt(ii,1)
        tab(l,2)=nt(ii,2)
        write(4,*)ne,l,nt(ii,1),nt(ii,2),mat(ne)
        write(7,*)l,x(nt(ii,2))-x(nt(ii,1)),y(nt(ii,2))-y(nt(ii,1)),
     1              z(nt(ii,2))-z(nt(ii,1))
      enddo
      ncount=l
      return
      end
C=======================================================================
C Determines edges corresponding to the surface element
C=======================================================================
      subroutine edge(m,j1,j2,j3)
      integer edgv(18000,3),tmp(3)
      common /dbase/edgv
      do ij=(6*m)-5,(6*m)
        if (j1.eq.edgv(ij,2)) then
```

```
      if (j2.eq.edgv(ij,3)) then
        tmp(1)=edgv(ij,1)
      elseif (j3.eq.edgv(ij,3)) then
        tmp(2)=edgv(ij,1)
      endif
    elseif (j2.eq.edgv(ij,2)) then
      if (j3.eq.edgv(ij,3)) then
        tmp(3)=edgv(ij,1)
      endif
    endif
  enddo
  write(8,*)m,j1,j2,j3,tmp
  return
  end
```

```
C=====================================================================
C.Computes the eigenvalues of an inhomogeneous cavity having arbitrary
C shape using edge elements.
C=====================================================================
      program fem
      integer edna(15000),gnn(15000,2),tab(6,3),sed(2000),edst(3000,2)
     1          ,mat(2500)
      character bit(6)*1
      real xyz(3000,3),a1(6,6),x(500),y(500),z(500),b1(6,6),eps(10)
      double precision a(700,700),b(700,700),alfr(700),alfi(700)
     1    ,beta(700)
        logical matz
      common /bank/edna,gnn,xyz,eps,mat
      common /mess/x,y,z
C.Reading in info
      write(6,*)'Number of edges'
      read(5,*)ned
      write(6,*)'Number of on-surface edges'
      read(5,*)nes
      write(6,*)'If inhomogeneous, enter 1'
      read(5,*)ivar
      if (ivar.eq.1) then
        write(6,*)'Number of different permittivities'
        read(5,*)nl
      endif
      open(1,file='eglob')
      open(2,file='edgy')
      open(3,file='enoddy')
      open(4,file='eig1')
      open(7,file='esurfed')
      read(1,*)(eps(i),i=1,nl)
      read(1,*)nn
      do i=1,nn
        read(3,*)k,x(i),y(i),z(i)
      enddo
        write(6,*)'Finished reading in nodes'
      read(1,*)nel
      do i=1,6*nel
        itemp=1+(i/6.)
        read(1,*)elm,edna(i),gnn(i,1),gnn(i,2),mat(itemp)
      enddo
      write(6,*)eps(mat(1)),eps(mat(47)),eps(mat(48)),eps(mat(74))
     write(6,*)'Finished reading in elements'
      do i=1,ned
        read(2,*)k,xyz(i,1),xyz(i,2),xyz(i,3)
      enddo
      write(6,*)'Finished reading in edges'
      read(7,*)(sed(i),i=1,nes)
      close(1)
      close(2)
      close(3)
      close(7)
C.The following program segment generates the FEM matrix and imposes
C the boundary condition for the on-surface nodes simultaneously.
C It compares the on-surface edges stored in the array 'sed' and
C stores the complement in 'a' and 'b'. It also keeps a record of
C the new edge numbers and their corresponding old edge numbers in
C the pointer array 'edst'.
      net=ned-nes
```

```
   do i=1,net
     do j=1,net
       a(i,j)=0.d0
       b(i,j)=0.d0
     enddo
    enddo
    write(6,*)'Generating FEM matrix'
    do i=1,3000
       edst(i,1)=0
       edst(i,2)=0
    enddo
    nptrx=1
    do i=1,nel
.Generate the matrix elements for the corresponding finite element
 and store the result in 'a1' and 'b1'. 'tab' is a pointer array
 which stores the node combinations and local and global edge nos.
      call crux(i,a1,b1,tab)
      do ij=1,6
        bit(ij)='0'
      enddo
.Run a check for the on-surface edges in the corresponding finite
 element. The character array 'bit' sets a flag for each on-surface
 edge in the finite element.
      do 100 j=1,6
        do ichk=1,nes
.If the element edge is an on-surface edge, set bit(i) ='1'
          if (tab(j,3).eq.sed(ichk)) then
            bit(j)='1'
            go to 100
          endif
        enddo
      enddo
.If bit(i)= '1', forget it. If bit(i) ='0', check to see whether
 the edge has occurred previously; if not, increment 'nptrx' and
 store the new edge no. and corresponding old edge no. in 'edst'
      do j=1,6
        if (bit(j).eq.'0') then
          if (edst(tab(j,3),2).ne.tab(j,3)) then
            edst(tab(j,3),1)=nptrx
            edst(tab(j,3),2)=tab(j,3)
            nptrx=nptrx+1
          endif
        endif
      enddo
.Only the edges inside the body are stored in the final array,
 'a' and 'b'.
      do j=1,6
        do k=1,6
          if ((bit(j).eq.'0').and.(bit(k).eq.'0')) then
            a(edst(tab(j,3),1),edst(tab(k,3),1))=
 1    a(edst(tab(j,3),1),edst(tab(k,3),1))+dble(a1(j,k))
            b(edst(tab(j,3),1),edst(tab(k,3),1))=
 1    b(edst(tab(j,3),1),edst(tab(k,3),1))+dble(b1(j,k))
          endif
        enddo
      enddo
    enddo
    write(6,*)'Finished FEM matrix generation'
    write(6,*)'Matrix is of order ',nptrx-1
```

```
   if (net.ne.(nptrx-1)) then
     write(6,*)'Error in matrix assembly'
     stop
   endif
   ir=0
    do i=1,nptrx-1
      do j=1,nptrx-1
        if ((b(i,j)-b(j,i)).gt..000001) then
          write(6,*)b(i,j),b(j,i),i,j
          ir=1
        endif
      enddo
    enddo
   if (ir.eq.0) then
     write(6,*)'Symmetry test passed'
   else
     write(6,*)'Symmetry test failed'
   endif
Solving the generalised eigenvalue problem: Ax=lambda*Bx
   eps1=.000001
   matz=.false.
   call qzhes(700,net,a,b,matz,z)
   write(6,*)'Step 1 done'
   call qzit(700,net,a,b,eps1,matz,z,ier)
   write(6,*)'Step 2 done'
  write(6,*)'error= ',ier
   call qzval(700,net,a,b,alfr,alfi,beta,matz,z)
   write(6,*)'Step 3 done'
   do i=1,nptrx-1
     alfr(i)=alfr(i)/beta(i)
   enddo
A bubble sort through the eigenvalue spectrum
   do i=1,net-1
     do j=net,i+1,-1
       if (alfr(j).lt.alfr(j-1)) then
         beta(j)=alfr(j)
         alfr(j)=alfr(j-1)
         alfr(j-1)=beta(j)
       endif
     enddo
       if (alfr(i).ge.0.) then
         write(4,*)sqrt(alfr(i))
       endif
   enddo
   close(4)
   stop
   end
=================================================================
   subroutine crux(l,a,b,tab)
   integer edna(15000),gnn(15000,2),tab(6,3),mat(2500)
   real xyz(3000,3),x(500),y(500),z(500),a(6,6),b(6,6),f(6,3),
1       g(6,3),tmp(3),eps(10)
   common /bank/edna,gnn,xyz,eps,mat
   common /mess/x,y,z
   common /local/sumx,sumy,sumz,xx,yy,zz,xy,yz,zx
   lv=6*(l-1)
   do j=1,6
     tab(j,1)=gnn(lv+j,1)
     tab(j,2)=gnn(lv+j,2)
```

```
      tab(j,3)=edna(lv+j)
    enddo
.Sorting the array 'tab' arranges it according to local node nos. so
 that the array looks like the one in file 'input'.
    call sort(tab)
.'calc' calculates some data corresponding to the element
    call calc(tab(1,1),tab(1,2),tab(2,2),tab(3,2))
.'volume' computes six times the volume of the tetrahedral element
    call volume(tab(1,3),tab(2,3),tab(3,3),vol)
    do j=1,6
     call cross(x(tab(7-j,1)),y(tab(7-j,1)),z(tab(7-j,1)),x(tab(
 1              7-j,2)),y(tab(7-j,2)),z(tab(7-j,2)),tmp)
.'bj' stores the length of the 'j' th edge.
      bj=sqrt((xyz(tab(j,3),1)**2)+(xyz(tab(j,3),2)**2)+(xyz
 1       (tab(j,3),3)**2))
.The constant vectors, 'f' and 'g', are computed and the result
 stored in 'f' and 'g', respectively.
      do k=1,3
        f(j,k)=bj*tmp(k)/vol
        g(j,k)=bj*xyz(tab(7-j,3),k)/vol
        if (j.eq.2) then
          g(j,k)=-1.*g(j,k)
        elseif (j.eq.5) then
          f(j,k)=-1.*f(j,k)
          g(j,k)=-1.*g(j,k)
        endif
      enddo
    enddo
    vol=vol/6.
    do j=1,6
      do k=1,6
.'a' contains the elemnts of the 'a' matrix in the main program.
  This equals the volume integral over curl W_i . curl W_j and
  is the same as 4.*g_i*g_j*V
        a(j,k)=4.*dot(j,k,g)*vol
.The function 'f1' computes the elements of the 'b' matrix in the
 main program. This is taken directly from the formulation and
 equals the volume integral over W_i . W_j
        b(j,k)=f1(j,k,f,g)*eps(mat(l))*vol
      enddo
    enddo
    return
    end
=======================================================================
    subroutine sort(tab)
    integer tab(6,3)
    nc=1
    do ik=4,2,-1
      if (nc.eq.1) then
        ij=6
      else
        ij=ik
      endif
      do ii=1,ij-1
        do j=ij,ii+1,-1
          if (tab(j,nc).lt.tab(j-1,nc)) then
            do jk=1,3
              call exchg(tab(j,jk),tab(j-1,jk))
            enddo
```

```
          endif
        enddo
      enddo
      nc=2
    enddo
    if (tab(5,2).lt.tab(4,2)) then
       do jk=1,3
         call exchg(tab(5,jk),tab(4,jk))
       enddo
      endif
    call exchg(tab(5,1),tab(5,2))
    return
    end
=======================================================================
    subroutine volume(j1,j2,j3,v)
    integer edna(15000),gnn(15000,2),mat(2500)
    real xyz(3000,3),b1(3),b2(3),b3(3),eps(10)
    common /bank/edna,gnn,xyz,eps,mat
    do it=1,3
      b1(it)=xyz(j1,it)
      b2(it)=xyz(j2,it)
      b3(it)=xyz(j3,it)
    enddo
    v=abs(b1(1)*((b2(2)*b3(3))-(b2(3)*b3(2)))-b1(2)*((b2(1)*b3
 1    (3))-(b2(3)*b3(1)))+b1(3)*((b2(1)*b3(2))-(b2(2)*b3(1))))
    return
    end
=======================================================================
    subroutine cross(x1,y1,z1,x2,y2,z2,tempo)
    real tempo(3)
    tempo(1)=y1*z2-y2*z1
    tempo(2)=z1*x2-z2*x1
    tempo(3)=x1*y2-x2*y1
    return
    end
=======================================================================
    subroutine calc(j1,j2,j3,j4)
    real x(500),y(500),z(500)
    common /mess/x,y,z
    common /local/sumx,sumy,sumz,xx,yy,zz,xy,yz,zx
    sumx=x(j1)+x(j2)+x(j3)+x(j4)
    sumy=y(j1)+y(j2)+y(j3)+y(j4)
    sumz=z(j1)+z(j2)+z(j3)+z(j4)
    xx=sumx*sumx+x(j1)*x(j1)+x(j2)*x(j2)+x(j3)*x(j3)+x(j4)*x(j4)
    yy=sumy*sumy+y(j1)*y(j1)+y(j2)*y(j2)+y(j3)*y(j3)+y(j4)*y(j4)
    zz=sumz*sumz+z(j1)*z(j1)+z(j2)*z(j2)+z(j3)*z(j3)+z(j4)*z(j4)
    xy=sumx*sumy+x(j1)*y(j1)+x(j2)*y(j2)+x(j3)*y(j3)+x(j4)*y(j4)
    yz=sumy*sumz+y(j1)*z(j1)+y(j2)*z(j2)+y(j3)*z(j3)+y(j4)*z(j4)
    zx=sumz*sumx+z(j1)*x(j1)+z(j2)*x(j2)+z(j3)*x(j3)+z(j4)*x(j4)
    return
    end
=======================================================================
    real function dot(j1,j2,vec)
    real vec(6,3)
    dot=(vec(j1,1)*vec(j2,1))+(vec(j1,2)*vec(j2,2))+(vec(j1,3)*vec
 1      (j2,3))
    return
    end
=======================================================================
```

```
      real function f1(j1,j2,f,g)
      real f(6,3),g(6,3),tmp1(3),tmp2(3)
      common /local/sumx,sumy,sumz,xx,yy,zz,xy,yz,zx
      call cross(f(j1,1),f(j1,2),f(j1,3),g(j2,1),g(j2,2),g(j2,3),tmp1)
      call cross(f(j2,1),f(j2,2),f(j2,3),g(j1,1),g(j1,2),g(j1,3),tmp2)
      term1=dot(j1,j2,f)
      term2=((tmp1(1)+tmp2(1))*sumx+(tmp1(2)+tmp2(2))*sumy+
     1       (tmp1(3)+tmp2(3))*sumz)/4.
      term3=(g(j1,2)*g(j2,2)+g(j1,3)*g(j2,3))*xx+(g(j1,1)*g(j2,1)+
     1       g(j1,3)*g(j2,3))*yy+(g(j1,1)*g(j2,1)+g(j1,2)*g(j2,2))*zz
     2      -(g(j1,1)*g(j2,2)+g(j1,2)*g(j2,1))*xy-(g(j1,1)*g(j2,3)+
     3       g(j1,3)*g(j2,1))*zx-(g(j1,2)*g(j2,3)+g(j1,3)*g(j2,2))*yz
      f1=term1+term2+(term3/20.)
      return
      end
=================================================================
      subroutine exchg(j1,j2)
      ntmp=j1
      j1=j2
      j2=ntmp
      return
      end
```

```
c=====================================================================
c rocesses a universal file obtained from IDEAS and stores the nodal info.
c hecks the surface nodes and imposes suitable boundary conditions on them.

c tores the node numbers and respective nodal coordinates in 'noddy'
c tores the element numbers and corresponding nodes in 'elno'
c torage limit: 4000 nodes, 25000 elements
c=====================================================================
      program unv_file_processor
      character string*80, yastrn*40, unv*20
      integer n1(25000,4),nn(50),nun(25000),bc(25000,3)
      integer count,tr(25000),tmp
      real x(4000),y(4000),z(4000)
      write(6,*)'1) Cavity problem 2) Scattering problem'
      read(5,*)ivar
      write(6,*)'Name of universal file ?'
      read(5,'(A)')unv
      open(1,file=unv)
      open(2,file='noddy')
      open(3,file='elno')
      open(8,file='surfed')
c Processing universal file for info on various parameters
      read(1,'(A)')string
      l=0
      if (string(4:6).eq.'151') then
        write(6,*)'Encountered header'
        go to 15
c Processing nodal info; data stored in 'noddy'
      elseif (string(4:6).eq.' 15') then
        read(1,'(A)')string
        if (string(5:6).ne.'-1') then
          write(2,*)string(7:10),' ',string(41:53),' ',string(54:66),' '
     1             ,string(67:79)
          l=l+1
          go to 5
        endif
        l1=l
        write(6,*)'There are ',l,' nodes.'
        write(8,*)l
c Processing element info; data stored in 'elno'
      elseif (string(4:6).eq.' 71') then
        read(1,'(A)')string
        if (string(5:6).ne.'-1') then
          read(1,'(A)')yastrn
          write(3,*)string(7:10),' ',yastrn(7:10),' ',yastrn(17:20),
     1             ' ',yastrn(27:30),' ',yastrn(37:40)
          l=l+1
          go to 10
        else
          write(6,*)'There are ',l,' elements.'
          write(8,*)l
          l2=l
        endif
      elseif (string(4:6).eq.'752') then
        ltmp=0
        nn(1)=0
        read(1,'(A)')string
        if (string(19:20).eq.' 0') then
          ltmp=ltmp+1
```

```
        read(string(57:60),'(I4)')numb
        nn(ltmp+1)=numb+nn(ltmp)
      elseif (string(5:6).eq.'-1') then
        write(6,*)'There are ',ltmp,' surface divisions.'
        go to 20
      elseif (string(1:1).eq.' ') then
        do i=1,4
          if (string((20*i-3):20*i).ne.'    ') then
            l=l+1
            read(string((20*i-3):20*i),'(I4)')tr(l)
          endif
        enddo
      endif
      go to 30
    endif
    go to 15
    close(1)
.Data from 'noddy' and 'elno' stored in
 n1,..,n4    - table of nodes making up the corresponding element
 x,y,z       - nodal coordinate table
    rewind 2
    rewind 3
    do i=1,l1
      read(2,*)na,x(i),y(i),z(i)
    enddo
    do i=1,l2
      read(3,*)nk,n1(i,1),n1(i,2),n1(i,3),n1(i,4)
    enddo
.Close 'noddy' and 'elno' for good
    close(2)
    ncnt=0
    do i=1,l2
      do j=1,3
        bc(i,j)=0
      enddo
    enddo
    if (ivar.eq.1) then
      do i=1,ltmp
        write(6,*)'Processing surface ',i,' for on-surface nodes.'
        do j=nn(i)+2,nn(i+1)
          if (x(tr(nn(i)+1)).eq.x(tr(j))) then
            flagx=0.
          else
            flagx=1.
          endif
          if (y(tr(nn(i)+1)).eq.y(tr(j))) then
            flagy=0.
          else
            flagy=1.
          endif
          if (z(tr(nn(i)+1)).eq.z(tr(j))) then
            flagz=0.
          else
            flagz=1.
          endif
        enddo
        do k=nn(i)+1,nn(i+1)
          if (flagx.eq.0.) then
            bc(k,1)=1
```

```
        elseif (flagy.eq.0.) then
          bc(k,2)=1
        elseif (flagz.eq.0.) then
          bc(k,3)=1
        endif
      enddo
    enddo
    do i=1,l-1
      do j=l,i+1,-1
        if (tr(j).lt.tr(j-1)) then
          call exchg(tr(j),tr(j-1))
          do k=1,3
            call exchg(bc(j,k),bc(j-1,k))
          enddo
        endif
      enddo
    enddo
    nuk=0.
    do i=1,l
      if (i.eq.1) then
        tmp=0
      else
        tmp=tr(i-1)
      endif
      if (tr(i).ne.tmp) then
        count=1
        if (tr(i+count).eq.tr(i)) then
          count=count+1
          go to 50
        endif
        nt1=1
        nt2=1
        nt3=1
        do j=1,count
          nt1=nt1*bc(i+j-1,1)
          nt2=nt2*bc(i+j-1,2)
          nt3=nt3*bc(i+j-1,3)
        enddo
        write(8,*)tr(i),nt1,nt2,nt3
        ncnt=ncnt+1
        nuk=nuk+(1-nt1)+(1-nt2)+(1-nt3)
      else
        go to 100
      endif
    enddo
    write(6,*)'No. of on-surface nodes = ',ncnt
    write(6,*)'No. of unknowns = ',3*l1-nuk
  else
    do i=1,ltmp
      write(8,*)i,nn(i+1)
    enddo
    do ii=1,ltmp
      do ik=1,12
        nun(ik)=0
      enddo
      do i=nn(ii)+1,nn(ii+1)
        do j=1,12
          do k=1,4
            if (tr(i).eq.n1(j,k)) then
```

```
              nun(j)=nun(j)+1
              bc(j,nun(j))=n1(j,k)
              if (nun(j).eq.3) then
                write(8,*)j,bc(j,1),bc(j,2),bc(j,3)
                ncnt=ncnt+1
                go to 200
              endif
            endif
          enddo
        enddo
      enddo
    enddo
    write(6,*)'There are ',ncnt,' surface elements.'
  endif
  close(3)
  close(8)
  stop
  end
=====================================================================
  subroutine exchg(j1,j2)
  ntmp=j1
  j1=j2
  j2=ntmp
  return
  end
```

```
C=====================================================================
C.This program computes the eigenvalues of a rectangular cavity
C using divergenceless nodal basis and tetrahedral elements.
C Storage limit: 500 nodes, 2000 elements, 300 surface nodes
C                1500 matrix size
C.To run program on Apollos:
C Need to run 'procn.obj' first; 'noddy', 'elno' and 'surfed' are created.
C ftn 'filename'
C bind 'filename'.bin /progs/math/naas/eispack.lib -b 'filename'.obj
C 'filename'.obj
C=====================================================================
      program fem_nodal_basis_3d
      logical matz
      integer elm(2000,4),surf(300),bc(300,3),tab(4)
      real a1(12,4),b1(12,4),x(500),y(500),z(500)
      double precision a(1500,1500),b(1500,1500),alfr(1500),alfi(1500)
     1    ,beta(1500)
      common /dbase/x,y,z,elm
      common /var/s,vol
      write(6,*)'No. of on-surface nodes'
      read(5,*)nsn
      write(6,*)'Penalty factor'
      read(5,*)s
      open(1,file='noddy')
      open(2,file='elno')
      open(3,file='surfed')
      open(4,file='eig')
      read(3,*)nn
      nn2=2*nn
      read(3,*)ne
      do i=1,nn
        read(1,*)nk,x(i),y(i),z(i)
      enddo
      do i=1,ne
        read(2,*)nk,elm(i,1),elm(i,2),elm(i,3),elm(i,4)
      enddo
      do i=1,nsn
        read(3,*)surf(i),bc(i,1),bc(i,2),bc(i,3)
      enddo
      write(6,*)'Finished reading in data'
      close(1)
      close(2)
      close(3)
C=====================================================================
      do iv=1,3
        do i=1,ne
          call crux(i,iv,a1,b1,tab)
          do j=1,12
            do k=1,4
              nt=(iv-1)*nn
              if (j.le.4) then
      a(tab(j),(nt+tab(k)))=a(tab(j),(nt+tab(k)))+dble(a1(j,k))
      b(tab(j),(nt+tab(k)))=b(tab(j),(nt+tab(k)))+dble(b1(j,k))
              elseif ((j.gt.4).and.(j.le.8)) then
      a((nn+tab(j-4)),(nt+tab(k)))=a((nn+tab(j-4)),(nt+tab(k)))
     1    +dble(a1(j,k))
      b((nn+tab(j-4)),(nt+tab(k)))=b((nn+tab(j-4)),(nt+tab(k)))
     1    +dble(b1(j,k))
              else
```

```
   a((nn2+tab(j-8)),(nt+tab(k)))=a((nn2+tab(j-8)),(nt+tab(k)))
1     +dble(a1(j,k))
   b((nn2+tab(j-8)),(nt+tab(k)))=b((nn2+tab(j-8)),(nt+tab(k)))
1     +dble(b1(j,k))
           endif
         enddo
       enddo
     enddo
   enddo
=====================================================================
   write(6,*)'Applying bc'
   do j=1,nsn
     if (bc(j,1).eq.0) then
       a(surf(j),surf(j))=-1.d0
     endif
     if (bc(j,2).eq.0) then
       a(nn+surf(j),nn+surf(j))=-1.d0
     endif
     if (bc(j,3).eq.0) then
       a(nn2+surf(j),nn2+surf(j))=-1.d0
     endif
   enddo
   write(6,*)'shifting rows'
   nptr=3*nn+1
   do j=1,(3*nn)
      if (float(j/15).eq.(j/15.)) then
        write(6,*)j,nptr
      endif
     if (a(j,j).eq.-1.d0) then
       nptr=nptr-1
       if (j.gt.nptr) go to 30
       if (a(nptr,nptr).ne.-1.d0) then
         do k=1,nptr
           call exchg(a(j,k),a(nptr,k))
           call exchg(b(j,k),b(nptr,k))
         enddo
         do k=1,nptr
           call exchg(a(k,j),a(k,nptr))
           call exchg(b(k,j),b(k,nptr))
         enddo
       else
         go to 20
       endif
     endif
   enddo
=====================================================================
   ir=0
   write(6,*)'Matrix generation complete; matrix of order ',nptr
   do i=1,nptr
     do j=1,nptr
       if ((a(i,j)-a(j,i)).gt..000001) then
         ir=1
         write(6,*)a(i,j),a(j,i),j,i
       endif
     enddo
   enddo
   if (ir.eq.1) then
     write(6,*)'Symmetry test failed for A'
   else
```

```
      write(6,*)'Symmetry test passed for A'
    endif
=======================================================================
    eps1=.00001
    matz=.false.
    call qzhes(1500,nptr,a,b,matz,z)
    write(6,*)'Step 1 done'
    call qzit(1500,nptr,a,b,eps1,matz,z,ier)
    write(6,*)'Step 2 done'
    write(6,*)'error= ',ier
    if (ier.ne.0) then
      write(6,*)'Erroneous computation; check FE matrices'
      stop
    endif
    call qzval(1500,nptr,a,b,alfr,alfi,beta,matz,z)
    write(6,*)'Step 3 done'
    do i=1,nptr
      alfr(i)=alfr(i)/beta(i)
    enddo
    do i=1,nptr-1
      do j=nptr,i+1,-1
        if (alfr(j).lt.alfr(j-1)) then
          beta(j)=alfr(j)
          alfr(j)=alfr(j-1)
          alfr(j-1)=beta(j)
        endif
      enddo
      if (alfr(i).gt.0.) then
      write(4,*)sqrt(alfr(i))
      endif
    enddo
    close(4)
    stop
    end
=======================================================================
    subroutine crux(l,iv,a,b,tab)
    integer elm(2000,4),tab(4)
    real a(12,4),b(12,4),x(500),y(500),z(500)
    common /dbase/x,y,z,elm
    common /var/s,vol
    do j=1,4
      tab(j)=elm(l,j)
    enddo
    call calc(iv,tab(1),tab(2),tab(3),tab(4))
    call volume(tab(1),tab(2),tab(3),tab(4),vol)
    do j=1,12
      do k=1,4
        call matx(iv,j,k,a(j,k),b(j,k))
      enddo
    enddo
    return
    end
=======================================================================
    subroutine volume(j1,j2,j3,j4,v)
    integer elm(2000,4)
    real x(500),y(500),z(500)
    common /dbase/x,y,z,elm
    v1=(x(j1)-x(j4))*((y(j2)-y(j4))*(z(j3)-z(j4))-(y(j3)-y(j4))*(z(j
   1     2)-z(j4)))
```

```
    v2=(y(j4)-y(j1))*((x(j2)-x(j4))*(z(j3)-z(j4))-(x(j3)-x(j4))*(z(j
 1     2)-z(j4)))
    v3=(z(j1)-z(j4))*((x(j2)-x(j4))*(y(j3)-y(j4))-(x(j3)-x(j4))*(y(j
 1     2)-y(j4)))
    v=abs(v1+v2+v3)/6.
    return
    end
=======================================================================
    subroutine calc(iv,j1,j2,j3,j4)
    integer elm(2000,4)
    real x(500),y(500),z(500),p(4),q(4),r(4)
    common /dbase/x,y,z,elm
    common /local/p,q,r
    p(1)=(y(j2)-y(j4))*(z(j3)-z(j4))-(y(j3)-y(j4))*(z(j2)-z(j4))
    p(2)=(y(j3)-y(j4))*(z(j1)-z(j4))-(y(j1)-y(j4))*(z(j3)-z(j4))
    p(3)=(y(j1)-y(j4))*(z(j2)-z(j4))-(y(j2)-y(j4))*(z(j1)-z(j4))
    p(4)=(y(j3)-y(j1))*(z(j2)-z(j1))-(y(j2)-y(j1))*(z(j3)-z(j1))
    q(1)=(z(j2)-z(j4))*(x(j3)-x(j4))-(z(j3)-z(j4))*(x(j2)-x(j4))
    q(2)=(z(j3)-z(j4))*(x(j1)-x(j4))-(z(j1)-z(j4))*(x(j3)-x(j4))
    q(3)=(z(j1)-z(j4))*(x(j2)-x(j4))-(z(j2)-z(j4))*(x(j1)-x(j4))
    q(4)=(z(j3)-z(j1))*(x(j2)-x(j1))-(z(j2)-z(j1))*(x(j3)-x(j1))
    r(1)=(y(j3)-y(j4))*(x(j2)-x(j4))-(y(j2)-y(j4))*(x(j3)-x(j4))
    r(2)=(y(j1)-y(j4))*(x(j3)-x(j4))-(y(j3)-y(j4))*(x(j1)-x(j4))
    r(3)=(y(j2)-y(j4))*(x(j1)-x(j4))-(y(j1)-y(j4))*(x(j2)-x(j4))
    r(4)=(y(j2)-y(j1))*(x(j3)-x(j1))-(y(j3)-y(j1))*(x(j2)-x(j1))
Checking routine follows; can be deleted if the user feels confident
enough of the matrix generation process
    t1=0.
    t2=0.
    t3=0.
    do ijk=1,4
      t1=t1+p(ijk)
      t2=t2+q(ijk)
      t3=t3+r(ijk)
    enddo
    if (t1.gt..000001) then
      write(6,*)t1,t2,t3
    endif
    return
    end
=======================================================================
    subroutine matx(m,j,k,fa,fb)
    real p(4),q(4),r(4)
    common /local/p,q,r
    common /var/s,vol
    tt=0.
    if (m.eq.1) then
      if (j.le.4) then
        t=(s*p(j)*p(k))+(q(j)*q(k))+(r(j)*r(k))
        if (j.ne.k) then
          tt=.05
        else
          tt=.1
        endif
      elseif ((j.gt.4).and.(j.le.8)) then
        jj=j-4
        t=(s*q(jj)*p(k))-(p(jj)*q(k))
      else
        jj=j-8
```

```
        t=(s*r(jj)*p(k))-(p(jj)*r(k))
      endif
    elseif (m.eq.2) then
      if (j.le.4) then
        t=(s*p(j)*q(k))-(q(j)*p(k))
      elseif ((j.gt.4).and.(j.le.8)) then
        jj=j-4
        t=(p(jj)*p(k))+(s*q(jj)*q(k))+(r(jj)*r(k))
        if (jj.ne.k) then
          tt=.05
        else
          tt=.1
        endif
      else
        jj=j-8
        t=(s*r(jj)*q(k))-(q(jj)*r(k))
      endif
    else
      if (j.le.4) then
        t=(s*p(j)*r(k))-(r(j)*p(k))
      elseif ((j.gt.4).and.(j.le.8)) then
        jj=j-4
        t=(s*q(jj)*r(k))-(r(jj)*q(k))
      else
        jj=j-8
        t=(p(jj)*p(k))+(q(jj)*q(k))+(s*r(jj)*r(k))
        if (jj.ne.k) then
          tt=.05
        else
          tt=.1
        endif
      endif
    endif
    fa=t/(36.*vol)
    fb=tt*vol
    return
    end
=================================================================
    subroutine exchg(x1,x2)
    tmp=x1
    x1=x2
    x2=tmp
    return
    end
```